Journal of Integrated
SCIENCE & TECHNOLOGY

Rapid Recover Map Reduce (RR-MR): Boosting failure recovery in Big Data applications

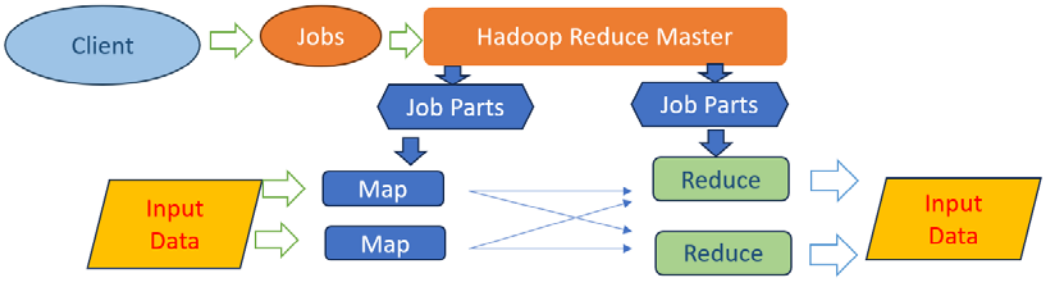
Sonika Anant Chorey,[1,2*] Neeraj Sahu[1]

[1]Computer Science and Engineering, G.H. Raisoni University, Amravati, India. [2]Prof. Ram Meghe Institute of Technology & Research, Badnera, Amravati, India.

Received on: 12-Oct-2023, Accepted and Published on: 14-Dec-2023

Article

ABSTRACT



The rapid growth of Big Data applications has brought forth unprecedented opportunities for insights and innovation, but it has also exposed the inherent vulnerabilities of data processing pipelines to failures. Hardware glitches, software anomalies, and network interruptions can disrupt the smooth execution of critical tasks, leading to extended downtimes, compromised reliability, and increased operational costs. In response to these challenges, we introduce Rapid Recover Map Reduce (RR-MR), an innovative framework designed to revolutionize failure recovery mechanisms within the context of Big Data applications. RR-MR addresses the shortcomings of conventional Map Reduce frameworks by presenting a novel approach to failure recovery that focuses on expeditious restoration of processing tasks. By leveraging advancements in distributed systems, fault tolerance, and parallel processing techniques, RR-MR introduces a multi-faceted strategy that enhances both the efficiency and reliability of recovery processes.

Keywords: Map Reduce; Fault tolerance; Checkpoint; Big data; Parallel computing

INTRODUCTION

In the fast-evolving landscape of Big Data applications, ensuring efficient and robust failure recovery is paramount to maintaining the reliability and performance of data processing tasks. This need has led to the innovation of Rapid Recover Map Reduce (RR-MR), a groundbreaking approach designed to elevate failure recovery mechanisms within the realm of Big Data processing.[1] In the realm of Big Data, where vast amounts of information are processed to extract valuable insights, the occurrence of failures is not uncommon. Hardware glitches, software bugs, network interruptions, and other unforeseen issues can disrupt the seamless execution of data processing jobs, leading to delays, data loss, and increased operational costs. Traditional Map Reduce frameworks, while capable, often struggle to swiftly recover from such failures, resulting in prolonged downtimes and hindered productivity.[2] RR-MR emerges as a game-changing solution to these challenges, heralding a new era of rapid and efficient failure recovery in Big Data applications. At its core, RR-MR re imagines the way failure recovery is approached within the Map Reduce paradigm. By harnessing innovative techniques drawn from distributed systems, fault tolerance, and parallel processing, RR-MR optimizes the recovery process, significantly reducing downtime and enabling applications to resume processing with minimal disruption.[3]

KEY FEATURES THAT DEFINE RR-MR INCLUDE:

- Enhanced Fault Detection
- Dynamic Task Redistribution
- Localized Recovery
- Adaptive Fault Tolerance

In the dynamic landscape of Big Data applications, the processing and analysis of large datasets have become a cornerstone of decision-making, innovation, and business growth. However, the sheer scale and complexity of these operations expose

*Corresponding Author: Sonika Anant Chorey
Email: sonikachorey@gmail.com

Cite as: *J. Integr. Sci. Technol., 2024, 12(3), 773.*
URN:NBN:sciencein.jist.2024.v12.773

them to a range of challenges, including hardware failures, software glitches, and network disruptions. The occurrence of such failures can lead to significant downtimes, data loss, and increased operational costs, underscoring the critical importance of efficient failure recovery mechanisms. Traditional Map Reduce frameworks, pioneered by Google and popularized by systems like Apache Hadoop, have long been the workhorses of Big Data processing. They provide a structured model for distributing processing tasks across a cluster of machines, enabling parallel computation and fault tolerance.[4] However, as the scale of Big Data applications has grown exponentially, the limitations of these traditional frameworks in terms of failure recovery have become increasingly evident. In many conventional Map Reduce implementations, failure recovery is a time-taking and resource intensive process. When a node fails during processing, the entire task needs to be restarted from scratch, causing delays and wasting precious computing resources.[5] Furthermore; the recovery process itself can introduce additional overhead, further prolonging the time taken to complete jobs. As a result, the efficiency and reliability of Big Data processing applications have been compromised, hindering their ability to deliver timely insights and meet the demands of rapidly evolving industries.[6]

The Hadoop Map Reduce workflow is a fundamental process in the Hadoop ecosystem

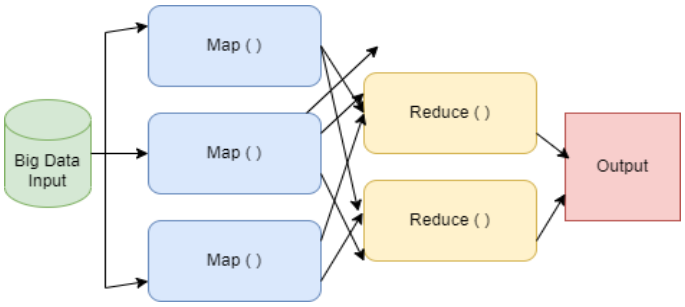


Figure 1: Map Reduce Work Flow

Designed for processing and analyzing large datasets in a distributed and parallel manner.[7] It follows a two-phase model consisting of the Map phase and the Reduce phase. The workflow is designed to harness the power of a cluster of commodity hardware to efficiently process vast amounts of data. Here's an overview of each phase and how they fit together:

- Map Phase
- Shuffling and Sorting
- Reduce Phase

Throughout the Map Reduce workflow, fault tolerance is achieved through mechanisms like task reassignment and data replication. If a map per or reducer task fails, it can be rescheduled to run on another node using the same input data. Additionally, intermediate data produced by map per is temporarily stored on local disks and replicated to ensure data availability even in the event of hardware failures. It's important to note that while the Map Reduce workflow provides a powerful way to process large datasets, it might not be the most efficient or suitable approach for all types of data processing tasks. As a result, the Hadoop ecosystem has expanded to include various tools and frameworks (such as Apache Spark) that build upon the Map Reduce model and offer additional features.

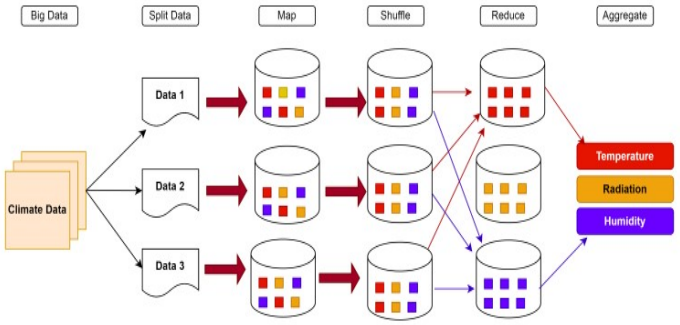


Figure 2. Example of map reduce work flow

Like in-memory processing and more flexible data processing paradigms.[8]

LITERATURE REVIEW

A consistent global checkpoint refers to a collection of states where no message is simultaneously recorded as received in one process and not yet sent in another process. Such checkpoints serve the purpose of facilitating rollbacks in the event of process failures. It is imperative that a consistent global checkpoint be acquired whenever any process initiates a checkpoint.[9] This paper introduces a checkpoint algorithm wherein the volume of information piggybacked on program messages remains independent of the number of mobile processes.

The algorithm aims to minimize the number of checkpoints based on two key assumptions: firstly, that a single consistent global checkpoint suffices for concurrent checkpoint initiations, and secondly, that a checkpoint is triggered at each handoff by mobile processes. Under these assumptions, the algorithm proves to be optimal among the generalizations of Chandy and Lamport's distributed snapshot algorithm.[10]

The contemporary banking sector holds significant importance in the lives of almost every individual, requiring interactions either in person or through online channels. However, in these interactions, both customers and banks are vulnerable to potential fraud schemes perpetrated by malicious actors. Various types of fraud, such as insurance fraud, credit card fraud, and accounting fraud, pose risks. Effectively detecting fraudulent activities becomes crucial to mitigate associated costs.[11] This paper focuses on addressing bank fraud detection through the application of data-mining techniques, including association, clustering, forecasting, and classification. The objective is to analyze customer data, unveiling patterns indicative of fraudulent behavior. Once these patterns are identified, enhancing banking processes with an additional layer of verification/authentication can be implemented.[12]

In recent years, IoT (Internet of Things) technology has found applications in the finance sector, leveraging generated data such as

real-time information from chattel mortgage supervision using GPS, sensors, network cameras, mobile devices, and more. This data is utilized to enhance the financial credit risk management of bank loans.[13] Financial credit risk stands out as one of the most significant challenges faced by commercial banks. However, as the volume of financial data grows exponentially from diverse sources like the Internet, mobile networks, and IoT, traditional statistical and neural network models may struggle to operate fairly or accurately in assessing credit risk with such varied data.[14]

Consequently, there is a pressing need to establish more robust risk prediction models employing artificial intelligence based on big data analytics. The goal is to predict default behaviors with improved accuracy and capacity. This article proposes a big data mining approach using Particle Swarm Optimization (PSO) based Back propagation (BP) neural network for financial risk management in commercial banks deploying IoT. The approach constructs a nonlinear parallel optimization model using Apache Spark and Hadoop HDFS techniques on datasets encompassing on-balance sheet and off-balance sheet items.[14]

Experimental results demonstrate that this parallel risk management model exhibits a fast convergence rate and formidable predictive capacity, proving efficient in identifying default behaviors. Furthermore, the distributed implementation on big data clusters significantly reduces the processing time for model training and testing.[15]

Time series data has become a focal point in contemporary research and is prevalent in various datasets. The prediction of time series phenomena is achieved through the exploration of time series data, allowing for the understanding of the developmental processes and patterns of socio-economic phenomena reflected over time.[16] This understanding aids in extrapolating and predicting the trends in their development. In the era of big data, there is an increasing emphasis on time series prediction, particularly in accurately forecasting trends.

This paper delves into various time series models, including autoregressive (AR) models, moving average (MA) models, and the ARIMA model, which combines both AR and MA components. As a fundamental application of time series prediction, accurate trend prediction is crucial.[17] The study applies the ARIMA model to predict risks in the National SME Stock Trading (New Third Board) in specific scenarios. Case studies demonstrate that our analysis results are generally consistent with the actual situation, significantly contributing to the prediction of financial risks.[18]

The occurrence of compute node failures has become a commonplace occurrence in many long-running and scalable MPI applications.[19] adhering to MPI standards and leveraging existing fault tolerance methods, we have devised a methodology that enables applications to withstand failures by implementing semi-coordinated checkpoints within the RADIC architecture. In pursuit of this goal, we have created the ULSC2-RADIC middleware, which segregates the application into independent MPI worlds, with each MPI world corresponding to a compute node. These independent worlds utilize the DMTCP checkpoint library within a semi-coordinated environment.[20]

Through experimentation with scientific applications and the NAS Parallel Benchmarks, we have assessed the overhead and functionality in the event of a node failure. Our evaluation focuses on comparing the computational cost of semi-coordinated checkpoints with coordinated checkpoints, providing insights into the efficiency and efficacy of our approach.[21]

As the scale of High-Performance Computing (HPC) clusters continues to grow, their increasing failure rates and energy consumption levels are emerging as serious design concerns.[22] Efficiently running systems at such large scales critically relies on deploying effective, practical methods for fault tolerance while having a good understanding of their respective performance and energy overheads. The most commonly used fault tolerance method is checkpoint/restart. Checkpoint scheduling policies, however, have been traditionally optimized and analyzed from one angle: application performance. In this work, we provide an extensive analysis of the performance, energy and I/O costs associated with a wide array of check pointing policies. We consider practical deployment issues and show that simple formulas can be used to accurately estimate wasted work in a system. We propose methods to optimize checkpoint scheduling for energy savings and evaluate the runtime-optimized and energy-optimized policies using simulations based on failure logs from 10 production HPC clusters. Our results show ample room for achieving high quality energy/performance tradeoffs when using methods that exploit characteristics of real world failures. We also analyze the impact of energy-optimized check pointing on the storage subsystem and identify policies that are optimal for I/O savings.[23]

FAULT TOLERANCE IN MAP REDUCE

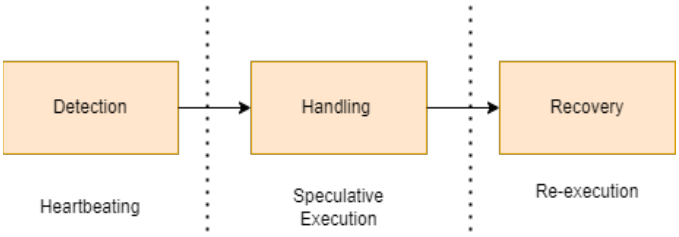


Figure 3. Fault Tolerance in Map Reduce

Fault tolerance is a crucial aspect of the Map Reduce framework, as it ensures that the processing of large-scale data continues uninterrupted even in the presence of hardware failures, software errors, or other disruptions. Map Reduce achieves fault tolerance through several mechanisms that maintain the reliability and completion of tasks:

a. Data Replication
b. Task Redundancy
c. Task Monitoring and Reassignment
d. Heartbeat Mechanism
e. Speculative Execution
f. Backup Task Attempts
g. Check pointing
h. Task Logs

FORMULATE THE CONCEPT OF RAPID RECOVERY MAP REDUCE. (FAR-MR)

In our quest to enhance the existing fault tolerance capabilities of the Map Reduce framework, we introduce an innovative approach known as Fast Recovery Map Reduce (FAR-MR). This study specifically focuses on the development of Dispersed FAR-MR, incorporating check pointing and a proactive push technique, both aimed at accelerating the recovery process in Map Reduce environments. In Dispersed FAR-MR, the primary mechanism is the utilization of distributed check pointing.[24] This involves the periodic marking of individual map tasks' progress as they complete, with the results stored in a distributed data repository. This approach ensures that in the event of a task or node failure, re-assigned tasks can swiftly access the most up-to-date progress information from the distributed storage. Consequently, tasks that are being recovered have the option to recommence computation from the point of failure, rather than initiating processing from scratch. Notably, the proactive nature of FAR-MR extends to the map output results. In this scenario, the map tasks, for example, M1 and M2, proactively transmit their output results to node S3, where the corresponding reduce task R1 for the same job is hosted.[25] This preemptive transmission of data ensures that the reduce tasks can benefit from incomplete map outputs generated by failed map tasks, should those map tasks not have completed their calculations.[26] This eliminates the necessity for recomposing specific data blocks, contributing to efficiency gains results are then proactively pushed from M1 and M2 to node S3, where reduce task R1 resides. In the unfortunate event of a failure in map task M1, the recovery mechanism seamlessly leverages the previously stored progress information to initiate recovery and efficiently resume processing. Through the strategic combination of distributed check pointing and proactive push techniques, Dispersed FAR-MR offers a promising pathway to significantly expedite failure recovery within the Map Reduce paradigm. This innovative approach holds the potential to enhance the efficiency, reliability, and overall performance of Big Data applications operating in challenging distributed environments.

FAR-MR IMPLEMENTATION

The implementation of FAR-MR involves several key components and strategies to enhance the efficiency and fault tolerance of Map Reduce jobs. Here's a high-level overview of the FAR-MR implementation:

Check pointing Technique:

FAR-MR utilizes check pointing to record the progress of map task computations. Checkpoints are created at strategic points, such as when a spill file is flushed to the local hard disk. Checkpoints capture essential details related to the computing job and task, including job_ID, task_ID, attempt_ID, spill_ID, and input_offset.

Distributed Check pointing:

FAR-MR goes beyond traditional check pointing by implementing a distributed check pointing mechanism. This means that checkpoints are not stored locally but are distributed across the cluster.

The distributed nature of checkpoints ensures redundancy and fault tolerance, allowing for recovery even if a node fails.

Proactive Pushing Mechanism:

- FAR-MR introduces a proactive pushing mechanism to optimize task recovery. Partial map output is proactively pushed to individual reducers, allowing them to be accessed and combined with partitions generated by the recovered task. This mechanism reduces the need for recompilation , as the recovered task can resume processing from the latest computing progress.

Handling Task and Node Failures

In the event of a single task failure, FAR-MR enables the recovered task to resume processing from the last known progress, avoiding redundant computations. For node failures, the distributed check pointing and proactive pushing mechanisms allow tasks to be rapidly recovered from any node in the cluster.

Performance Comparison

FAR-MR evaluates its performance by comparing it with traditional Hadoop Map Reduce, particularly focusing on scenarios involving task failure recovery. The evaluation demonstrates significant performance improvements with FAR-MR, showcasing its effectiveness in reducing computing time during failure recovery.

DATASET-SPECIFIC NODE FAILURE HANDLING

FAR-MR adapts its approach based on the size of the dataset and the number of data blocks. It considers the number of node failures that can be handled for each dataset size, tailoring its fault tolerance mechanisms accordingly.

We have incorporated the designed distributed check pointing into FAR-MR depends on Hadoop Map Reduce, and proactive push mechanism. We employ a Redis cluster to save the checkpoints in the context of distributed check pointing. The distributed memory to enable both locally and remotely quick failure recoveries. Redis stores checkpoint Rows as strings data structures with attempt ID spill ID input of set as the value and job ID task ID as the unique key. The record is written as key, value > to the Redis cluster whenever a new checkpoint is generated. If a record with the resemblance key already exists in Redis, the new value replaces it. We have devised an independent process that the map job will invoke for the proactive push mechanism at each spill occurrence.[27] before sending each part to the node responsible for the relevant reducers, this procedure initially divides the spill flow into multiple partitions, each aligning with distinct reducers. It's important to note that this push process operates independently of the system and does not effect the current work map and decrease tasks, ensuring that their completion timeline remains unaffected.[28] To facilitate the prompt initiation of data transfer from map tasks to reduce tasks in support of the push mechanism, it is imperative for the scheduler to initiate reduction jobs early. In the case of FAR-MR, we have configured the necessary settings to initiate reduce job scheduling as soon as the map output becomes available.

EVALUATION OF PERFORMANCE

We established a computing cluster comprising 24 nodes to conduct a performance evaluation comparing FAR-MR and Hadoop Map Reduce, with a specific focus on examining FAR-MR's performance at the context of failure recovery. This cluster is composed of 223 slave virtual machine nodes, each equipped with

4 Central Processing Unit cores and 16 Giga Byte of memory, along with 2 master nodes. To support the distributed checkpoint storage, FAR-MR deploys a Redis cluster. Both FAR-MR and Hadoop Map Reduce were tasked with executing Text analysis task to evaluate their respective examine in the context of fault recovery. They incrementally increased the size of the input dataset from 256 MB to 2 GB, with each data block or chunk being 256 MB in size. In the case of FAR-MR and Hadoop Map Reduce, we intentionally created scenarios involving Work disruption and system node breakdown within the cluster. Subsequently, we measured the period required to finish the Word Count jobs during the fault recovery process. To induce map task failures during job execution, we introduced erroneous records into the input data. These flawed records were strategically inserted into the data blocks, positioned at distances of 96%, 62%, and 35% from the beginning of each block. To mimic node failure scenarios, we alter the node management. We evaluate the system's performance under conditions of up to four concurrent node failures. To calculate the average performance result for each measurement, we execute the Word Count tasks four times with identical settings, ensuring the cache is cleared between each run. For performance comparison, we also conduct equivalent tests using both FAR-MR and Hadoop Map Reduce.

Failure of Task

In the context of task failure recovery, we conducted a performance comparison between FAR-MR and Hadoop Map Reduce while evaluating the completion time for Word Count computing jobs with 1, 2, and 3 failures. Our findings reveal that FAR-MR significantly outperforms Hadoop MapReduce, particularly in the realm of failure recovery.

In instances of a single task failure, FAR-MR demonstrates an impressive 55% performance improvement compared to the original Hadoop Map Reduce across all datasets. This superiority is attributed to FAR-MR's ability to retain the latest computing progress, enabling recovered tasks to resume from that specific point. In contrast, Hadoop Map Reduce necessitates the re-computation of data blocks from the beginning in the event of task failure. FAR-MR further enhances efficiency by proactively transmitting partial map output to individual reducers. These outputs can then be accessed by the reducers and seamlessly integrated with partitions generated by the recovered task. In cases of single task failure, the break point data block is approximately 95%, allowing the recovered task in FAR-MR to compute only the remaining 5% during the map phase, resulting in a significant reduction in computing time.

The evaluation results indicate that FAR-MR continues to yield performance improvements over Hadoop Map Reduce as the number of task failures increases. This can be attributed to FAR-MR's consistent ability to shorten computing times with each failure occurrence. The cumulative effect of multiple failures results in even greater performance improvements achieved by FAR-MR compared to scenarios involving a single failure.

Failure Node

Node failures occur primarily in larger datasets, as smaller datasets are distributed across a restricted number of nodes in HDFS. Specifically, for a 256 MB dataset comprising a single data block, we exclusively assess performance in the context of a single node failure. In the case of a 512 MB dataset with two data blocks, performance evaluation extends to scenarios involving up to 2 node failures. This pattern continues, with performance measurements accommodating up to 4 node failures for 1 GB and 2 GB datasets, respectively.

This notable enhancement is primarily attributed to the fact that FAR-MR ensures the map task continually maintains the most up-to-date computational progress. Consequently, when a task needs to recover, it can resume its operation from this point. In contrast, with Hadoop Map Reduce, the recovered task must recomputed the entire data block from scratch. Furthermore, FAR-MR employs a proactive approach by transmitting partial map output to individual reducers. This design allows these outputs to be readily accessible by the reducers and seamlessly integrated with the partitions generated by the recovered task.

Our innovative FAR-MR approach leverages the check pointing technique to log computing progress, expediting the recovery of Map Reduce jobs in a manner akin to existing strategies. However, FAR-MR distinguishes itself by incorporating distributed check pointing and proactive pushing mechanisms, setting it apart from conventional strategies. These novel mechanisms not only facilitate swift recovery from both task and node failures in Map Reduce jobs but also enable the rapid recovery of tasks from any node within the cluster.

CONCLUSION

The current failure tolerance strategy within Hadoop Map Reduce imposes need for computing entire data blocks when dealing with recovered tasks, resulting in a substantial performance penalty and resource inefficiency during the recovery process. To tackle this challenge, this research paper introduces Fast Recovery Map Reduce (FAR-MR), which focuses on implementing a failure tolerance approach for big data applications that allows tasks to be resumed from where they left off. FAR-MR leverages check pointing and distributed storage technologies to enable recovered tasks to recommence their computations from their previous noted growth. The suggested FAR-MR system is put into practice and assessed on a server cluster comprising 23 nodes. It is evaluated in scenarios involving both task fault recovery and node fault recovery. The efficiency assessment clearly demonstrates that, in comparison to Hadoop Map Reduce, FAR-MR achieves substantial improvements, with efficiency enhancements of up to 62% in the case of task fault recovery and 45% in the case of node failure recovery. Moreover, the efficiency advantage of FAR-MR over Hadoop Map Reduce becomes more pronounced as the number of task failures increases. These findings underscore the potential of FAR-MR to deliver enhanced efficiency when confronted with failures in the processing of big data tasks.

ACKNOWLEDGMENTS

The research was not funded by any specific grants from public, commercial, or not-for-profit sectors. The authors express their gratitude to G. H. Raisoni University, Amravati for their support and provision of all the necessary lab facilities to conduct this research.

CONFLICT OF INTEREST STATEMENT

Authores do not have any conflict of interest in publishing of this work.

REFERENCES AND NOTES

1. I. Ud Din, M. Guizani, S. Hassan, et al. The Internet of Things: A Review of Enabled Technologies and Future Challenges. *IEEE Access* **2019**, 7, 7606–7640.
2. T.A. Ahanger, A. Aljumah. Internet of things: A comprehensive study of security issues and defense mechanisms. *IEEE Access* **2019**, 7, 11020–11028.
3. T. Qiu, J. Liu, W. Si, D.O. Wu. Robustness Optimization Scheme with Multi-Population Co-Evolution for Scale-Free Wireless Sensor Networks. *IEEE/ACM Trans. Netw.* **2019**, 27 (3), 1028–1042.
4. R. Wang, C. Yu, J. Wang. Construction of Supply Chain Financial Risk Management Mode Based on Internet of Things. *IEEE Access* **2019**, 7, 110323–110332.
5. C. Shepherd, F.A.P. Petitcolas, R.N. Akram, K. Markantonakis. An exploratory analysis of the security risks of the internet of things in finance. *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)* **2017**, 10442 LNCS, 164–179.
6. Q. Zhang, P. Shi, A. Zeng, Y. Ma, X. Yuan. Dynamic control analysis of intensified extractive distillation process with vapor recompression. *Sep. Purif. Technol.* **2020**, 233, 116016.
7. C. Amornbunchornvej, T.Y. Berger-Wolf. Mining and modeling complex leadership–followership dynamics of movement data. *Soc. Netw. Anal. Min.* **2019**, 9 (1), 1–17.
8. N. Losada, G. Bosilca, A. Bouteiller, P. González, M.J. Martín. Local rollback for resilient MPI applications with application-level checkpointing and message logging. *Futur. Gener. Comput. Syst.* **2019**, 91, 450–464.
9. N. Losada, P. González, M.J. Martín, et al. Fault tolerance of MPI applications in exascale systems: The ULFM solution. *Futur. Gener. Comput. Syst.* **2020**, 106, 467–481.
10. D. Zhong, X. Luo, A. Bouteiller, G. Bosilca. Runtime level failure detection and propagation in HPC systems. In *ACM International Conference Proceeding Series*; **2019**; pp 1–11.
11. X. Tang, J. Zhai, B. Yu, et al. An efficient in-memory checkpoint method and its practice on fault-tolerant HPL. *IEEE Trans. Parallel Distrib. Syst.* **2018**, 29 (4), 758–771.
12. Z. Liu, T. Liu, J. Han, et al. Signal Model-Based Fault Coding for Diagnostics and Prognostics of Analog Electronic Circuits. *IEEE Trans. Ind. Electron.* **2017**, 64 (1), 605–614.
13. Z. Gao, C. Cecati, S.X. Ding. A survey of fault diagnosis and fault-tolerant techniques-part I: Fault diagnosis with model-based and signal-based approaches. *IEEE Trans. Ind. Electron.* **2015**, 62 (6), 3757–3767.
14. H. Miao, B. Li, C. Sun, J. Liu. Joint Learning of Degradation Assessment and RUL Prediction for Aeroengines via Dual-Task Deep LSTM Networks. *IEEE Trans. Ind. Informatics* **2019**, 15 (9), 5023–5032.
15. R. Iqbal, T. Maniak, F. Doctor, C. Karyotis. Fault Detection and Isolation in Industrial Processes Using Deep Learning Approaches. *IEEE Trans. Ind. Informatics* **2019**, 15 (5), 3077–3084.
16. D. Binu, B.S. Kariyappa. RideNN: A New Rider Optimization Algorithm-Based Neural Network for Fault Diagnosis in Analog Circuits. *IEEE Trans. Instrum. Meas.* **2019**, 68 (1), 2–26.
17. L. Ramalho, I. Freire, C. Lu, M. Berg, A. Klautau. Improved LPC-based fronthaul compression with high rate adaptation resolution. *IEEE Commun. Lett.* **2018**, 22 (3), 458–461.
18. Z. Chen, J. Xu, J. Tang, K. Kwiat, C. Kamhoua. G-Storm: GPU-enabled high-throughput online data processing in Storm. In *Proceedings - 2015 IEEE International Conference on Big Data, IEEE Big Data 2015*; **2015**; pp 307–312.
19. G. Levitin, L. Xing, Y. Dai. Optimal backup frequency in system with random repair time. *Reliab. Eng. Syst. Saf.* **2015**, 144, 12–22.
20. R. Jhawar, V. Piuri. Fault Tolerance and Resilience in Cloud Computing Environments. In *Computer and Information Security Handbook*; Elsevier, **2017**; pp 165–181.
21. Y. Sharma, B. Javadi, W. Si, D. Sun. Reliability and energy efficiency in cloud computing systems: Survey and taxonomy. *J. Netw. Comput. Appl.* **2016**, 74, 66–85.
22. S. Chorey, N. Sahu. Failure recovery model in big data using the checkpoint approach. *J. Integr. Sci. Technol.* **2023**, 11 (4), 564.
23. P. Chorey, N. Sahu. Enhancing efficiency and scalability in Blockchain Consensus algorithms: The role of Checkpoint approach. *J. Integr. Sci. Technol.* **2024**, 12 (1), 706.
24. K. Mohror, A. Moody, G. Bronevetsky, B.R. De Supinski. Detailed modeling and evaluation of a scalable multilevel checkpointing system. *IEEE Trans. Parallel Distrib. Syst.* **2014**, 25 (9), 2255–2263.
25. T. Ozaki, T. Dohi, H. Okamura, N. Kaio. Distribution-free checkpoint placement algorithms based on min-max principle. *IEEE Trans. Dependable Secur. Comput.* **2006**, 3 (2), 130–140.
26. H. Wang, L.S. Peh, E. Koukoumidis, S. Tao, M.C. Chan. Meteor shower: A reliable stream processing system for commodity data centers. In *Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium, IPDPS 2012*; **2012**; pp 1180–1191.
27. C. Xu, M. Holzemer, M. Kaul, J. Soto, V. Markl. On Fault Tolerance for Distributed Iterative Dataflow Processing. *IEEE Trans. Knowl. Data Eng.* **2017**, 29 (8), 1709–1722.
28. S. Jayasekara, S. Karunasekera, A. Harwood. Enhancing the Scalability and Performance of Iterative Graph Algorithms on Apache Storm. In *Proceedings - 2018 IEEE International Conference on Big Data, Big Data 2018*; **2018**; pp 3863–3872.
29. S. Chorey, N. Sahu. Securing the Crash Failures and Accidentally Destroyed of Large Data using Checkpoint Approach. *13th Int. Conf. Adv. Comput. Control. Telecommun. Technol. ACT 2022* **2022**, 8, 146–152.